Projet Informatique et Sciences du Numériques

I. Présentation du projet

Cette année, en Terminale S, j'ai pris part à la spécialité ISN au lycée Louis Marchal. Nous avons d'abord appris le langage python grâce à différents TP (bataille navale, morpion...), et avons créé notre propre site Web : luc.varoqui.org

Nous avons ensuite du nous lancer dans un projet de groupe, nous avons choisi, avec mes deux acolytes Lucas Caletti et Robin Troesch, de recréer un Space Invaders en hommage au jeu vidéo mythique des bornes d'arcades. Le principe du jeu est simple, le joueur contrôle un vaisseau et doit tuer tous les envahisseurs avant que ces derniers n'atteignent le bas de l'écran. Le joueur gagne des points en fonction du nombre de monstres qu'il touche (certains monstres possèdent un bouclier, le détruire rapportera un point). Un menu, un système de meilleur score et d'évolution de la difficulté sont également présents.

Le jeu est organisé comme ceci à l'intérieur de son dossier :

- Space_Invaders.py qui est le menu est donc le fichier à lancer pour démarrer le jeu
- Main_Game.py qui est le jeu en lui-même, lancé par Space_Invaders.py
- Images où sont stockés toutes nos images (boutons, monstres, joueur...)
- Sons où sont stockés tous les sons utilisés dans notre jeu
- Config.txt dans lequel sont stockés le meilleur score et les différents paramètres utilisateur

II. Mode d'emploi

Afin de pouvoir lancer le jeu vous aurez besoin de :

- python 3.2
- pygame
- l'intégralité du dossier du jeu

Le lancement du jeu se fait en lançant le fichier *Space_Invaders.py*, ensuite vous pourrez naviguer dans le menu grâce à la souris :

Bruitages ON

Bruitages OFF

Musique ON

Musique OFF

Une fois en jeu :

Touche	Action
Barre d'espace	Tirer
Q	Se déplacer à gauche
D	Se déplacer à droite

III. Répartition des tâches

Lucas Caletti	Lucas s'est occupé de toute la création du menu : • Fonctions permettant de changer l'état des boutons (au repos, survolés, cliqués) • Disposition des boutons • Gestion du changement de fenêtre (menu ↔ jeu)
Robin Troesch	• Gestion des monstres dans le jeu (création, déplacements, affichage et explosions de ces derniers) • Gestion des scores et du stockage du meilleur score.
Moi	• Gestion du vaisseau déplacements • Fonctions permettant de tirer un projectile • « Première » version de la gestion des collisions entre projectiles et vaisseau que Robin a modifié par la suite afin de pouvoir intégrer les monstres à bouclier. • Fonction *wait* permettant de mettre en pause le jeu • Gestion des sons • Conditions de victoires et de défaites. • Création de toutes les images du jeu

IV. Vue globale du programme

V. Stratégies et principes du programme

Afin de pouvoir gérer plusieurs éléments identiques (monstres, projectiles,…) de manière simultanée, Robin et moi avons choisi d'utiliser des *Classes*, une classe est un « moule » à partir duquel il est possible de créer des objets. Elles possèdent des propriétés et des méthodes.

Nativement, pygame incorpore la classe *Sprite*, qui possède des propriétés et des méthodes adaptées aux besoins d'un jeu vidéo (création de groupes, gestion d'un *Update*…). Nous avons créé des classes héritant de la classe Sprite, c'est-à-dire bénéficiant automatiquement des propriétés et méthodes de cette dernière. Nous y avons ajouté des méthodes spécifiques à chaque objet et avons ainsi créé nos propres classes : Player, Bullet, Invaders,…

Il est à noter qu'un objet n'est pas forcément présent sur l'interface graphique.

Pour créer un nouvel objet, il suffit de taper : nom_de_l'objet = nom_de_la_classe()

Le grand avantage est que l'on peut créer une infinité d'objets de même nom. A leur création nous plaçons les objets dans des listes de Sprites : all_sprites_list, bullet_list… Parcourir une liste permet d'interagir avec tous les objets de cette liste en même temps.

Un exemple de classe, « Class Bullet » :

```
def __init__(self):
    super().__init__()              #Permet de lancer le constructeur de la classe Sprite
    self.image = bullet_image  #On définit l'image assigné à un projectile
    self.rect = Rect(0,0,4,10)              #On définit le rectangle du Sprite
    self.rect.x = player.rect.x + 45        #On place le bullet au-dessus du Vaisseau
    self.rect.y = player.rect.y
    self.add(bullet_list, all_sprites_list) #On ajoute le bullet aux listes nécessaires
def update(self):
    self.rect.y -= 8        #Fait monter le projectile de 8 pixels

    if self.rect.y < -10:  #Enlève le missile si il dépasse le cadre  de l'écran
      self.remove()
      self.kill()
```

- Le *__init__(self)* correspond au constructeur, elle est appelé lorsque l'on instancie un nouvel objet, ici un projectiles.
- La méthode update(self) sera exécutée à chaque passage dans la boucle principale. On "update" tous les objets d'un seul coup grâce à : all_sprites_list.update()
  Pour un projectile, le update consiste à faire monter ce dernier de 8 pixels. De plus, vérifie que le projectile n'est pas en dehors de l'écran, s'il l'est on le supprime dans un souci d'économie de ressources systèmes.

Gérer le contrôle du personnage :

```
keys = pygame.key.get_pressed()          #Récupère la liste de toutes les touches pressées par l'utilisateur

#Action en fonction des touches pressées
if keys[K_a] and keys[K_SPACE]:
        player.rect.x -= speed_vaisseau
        shoot_again = placebullet(shoot_again)

elif keys[K_d] and keys[K_SPACE]:
        player.rect.x += speed_vaisseau
        shoot_again = placebullet(shoot_again)

elif keys[K_a]:
        player.rect.x -= speed_vaisseau

elif keys[K_d]:
        player.rect.x += speed_vaisseau

elif keys[K_SPACE]:
        shoot_again = placebullet(shoot_again)
```

Afin de se déplacer et de tirer, le joueur utilise les touches du clavier Q,D et Espace. Afin de récupérer les touches sur lesquels appui le joueur et ensuite permettre une action, on commence par récupérer une liste de booléen comportant toute les touches du clavier, on regarde ensuite quelle touches sont appuyés et l'on agit en conséquence. Le grand avantage de cette méthode par rapport à celle des "EVENT" est que l'on peut gérer l'appui simultané sur les touches. On peut donc se déplacer et tirer en même temps.

On peut noter que pour récupérer la position de la touche Q on regarde dans l'index [K_a] car pygame voit notre clavier comme un QWERTY.

Gérer la création des projectiles, la fonction placebullet:

```
def placebullet(delay):
    bullet = Bullet()
    return shoot_delay
return delay
```

Afin d'éviter que le joueur puisse tirer en continu, j'ai mis en place la variable shoot_again et la constante shoot_delay. shoot_again se décrémente à chaque passage dans la boucle principale, le joueur ne peut tirer que quand cette variable est inférieure à zéro. Si le joueur est autorisé à tirer, on créer un objet Bullet et l'on remet la variable shoot_again à la valeur shoot_delay (j'ai choisi de mettre cette dernière dans une constante afin de pouvoir l'adapter par la suite, pour changer la cadence de tir du vaisseau).

Gestion des projectiles, la fonction bullet_traitement :

```
for bullet in bullet_list:        #Parcours de la liste de bullet

    monster_hit_list = pygame.sprite.spritecollide(bullet, monster_list, False)
    for monster in monster_hit_list:
        bullet.kill()
```

Afin de traiter les projectiles, on parcourt la liste des Bullet, et l'on traite ces derniers séparément. Grâce à Pygame j'ai pu détecter une collision entre un Sprite en particulier (en l'occurrence un Bullet) et n'importe quel Sprite d'une Sprite List (ici les Invaders). La fonction retourne alors une autre Sprite List comportant tous les monstres en collisions avec le Bullet (la plupart du temps il n'y aura qu'un monstre dans cette liste, le cas d'un Bullet touchant deux monstres simultanément étant quasi impossible). On parcourt cette liste et l'on détruit le bullet qui est entré en collision avec un monstre, bien sûr si le bullet n'est pas entré en collision avec un monstre la liste sera vide et il ne sera pas détruit.

Robin a ensuite modifié ma fonction afin d'ajouter un système de point de vie, en effet au début on supprimait directement le monstre et le projectile sans prise en compte des monstres avec boucliers.

La gestion des sons :

Pour les bruitages de tirs du vaisseau nous avons utilisé le module Mixer de pygame.

```
laser = pygame.mixer.Sound("Sons/laser.ogg")  #Création de l'objet "laser"
laser.play()    #Lecture du son de tir
```

Pour la lecture de la musique le module Music était plus adapté, il permet en effet de créer des listes de lectures et donc d'enchainer automatiquement les musiques (bien que dans notre projet nous n'en diffusons qu'une seule).

```
pygame.mixer.music.load("Sons/unity.wav")     #On charge le morceau dans la Playlist
pygame.mixer.music.play()       #On lance la lecture de la Playlist
```

Nous avons également intégré la possibilité de couper, soit les sons, soit les bruitages (grâce aux deux boutons en bas à droite du menu). Pour ce faire, on stock l'information dans un fichier texte : Config.txt, on peut ensuite lire et modifié ce fichier en fonction des envies de l'utilisateur. L'information est stockée sur 2 bits, par exemple un « 00 » dans le fichier indiquera que ni les bruitages ni les sons ne doivent être joués, alors qu'un « 11 » indiqueront qu'ils devront l'être. Le premier bit correspond à la variable Music et le deuxième à Sound_Effect. Dans le jeu, la configuration du son se fait au lancement de la fenêtre grâce à la fonction config_son.

```
def config_son():
    fichier_config = open("Config.txt", "r")          #On ouvre le fichier de configuration
    config = fichier_config.read()            #On stock son contenu dans une chaîne de caractère
    if config == "00":       #On agit en fonction de cette chaîne de caractère
        pygame.mixer.music.set_volume(0)          #Permet de mettre le volume de la musique à 0
        Sound_Effect = 0              #Permet de ne pas jouer les bruitages (voir ci-dessous)
    elif config == "11":
        pygame.mixer.music.set_volume(1)
        Sound_Effect = 1
    elif config == "01":
        pygame.mixer.music.set_volume(0)
        Sound_Effect = 1
    elif config == "10":
        pygame.mixer.music.set_volume(1)
        Sound_Effect = 0
    return Sound_Effect
```

La variable Sound_Effect est utilisé dans la fonction place_bullet et permet de décider ou non de la lecture du bruitage.

```
if Sound_Effect == 1:
        laser.stop()
        laser.play()    #Lecture du son de tir
```

VI. Pistes d'améliorations

Nous avons réussi à répondre à notre cahier des charges minimal, cependant beaucoup d'améliorations peuvent être imaginée :

- Création d'autre niveau
- Création d'un mode « Multijoueur » avec 2 vaisseaux
- Augmentation de la difficulté du jeu par l'ajout de tirs venant des envahisseurs
- Création d'un système de meilleur score avec le nom de la personne

VII. Conclusion

L'option ISN m'a particulièrement plu tout au long de l'année, j'ai appris beaucoup de choses sur la programmation en python mais aussi sur d'autres sujets touchant à l'informatique, la création de mon propre site web, l'encodage des caractères et des images... Voulant devenir ingénieur en informatique cette option m'a conforté dans mon choix.

C'était l'un de mes rêves d'enfant de pouvoir créer m'on propre jeu-vidéo, et ce projet a été pour moi la preuve qu'avec l'aide de mes deux acolytes nous en étions capable. Le fait de travailler en groupe a été un vrai plus, le point de vue des autres étant très intéressant et permettant de faire avancer les choses. Aboutir à un résultat concret nous a fait particulièrement plaisir !

VIII. Annexe

Les codes sources des deux fichiers Python ainsi qu'une vidéo de démonstration sont disponibles sur mon site dans la section projet. L'adresse du site : luc.varoqui.org